Internship report

Using a chatbot to get variables for a Kubernetes cluster script

Sarah Greb
S1083409
Artificial Intelligent M2
Radboud University

Internship period:
February 1, 2023 - April 25, 2023

Internal supervisor:
Johan Kwisthout
Associate professor
Artificial Intelligent Radboud

Internal assessor:
Cristian Tejedor Garcia
Post-doctoral researcher
Centre for Language and Speech Technology

Table of Content

1 Introduction

The internship was completed in a German IT company,.The duration of the internship was in the third period of the academic year 2022-2023 from 01.02.2023-25.04.2023 with a total of 450 h.

The motivation of the internship was to explore the feasibility of building a customized chatbot for getting variables for a Kubernetes script, as a potential internal tool for the company. In more detail, the chatbot would help non-technical coworkers to make a YAML file for a Kubernetes Onboarding. The Onboarding was done on a platform called Kubernetes as a Services (KaaS). With the help of the chatbot less human resource may be needed. The development of the chatbot was done using Rasa.

Four learning goals are expected to be achieved. The first two learning goals of the internship are working with real life data and using a state-of-the-art program. Especially the work with real-life data will give a lot of important lessons. As seen in previous projects in the Master's program, real-life data is behaving differently to a clean dataset. With this challenge the learned theoretical method from the master can be applied in practice. The third learning goal is to integrate a state-of-the-art method into a functional workflow, as that is more challenging than an AI on its own. Also, the last learning goal is to get to know the everyday work life inside the industry.

The expected results of the internship are to have a functional chatbot that can extract the needed variables for the script used in KaaS. Also, to get an understanding of the work life inside the industry and how it is fitting for the student. Another result is to learn how to apply state- of-the-art methods to new problems.

The activities of the internship are described. The activities will be stated chronically and in topics. The topics are: decision of the chatbot platform, getting to know the chatbot platform, building the first version of the chatbot, deployment of the chatbot, improvement and finetuning the chatbot, evaluation of the chatbot. Lastly the evaluation of the internship will be done. Here the report will reflect on the learning goals, how they were met, the process compared to the planned schedule, the supervision process, and a final summary. The report ends with an appendix with the time logs and examples of the chatbot.

2 Activities

2.1 Research

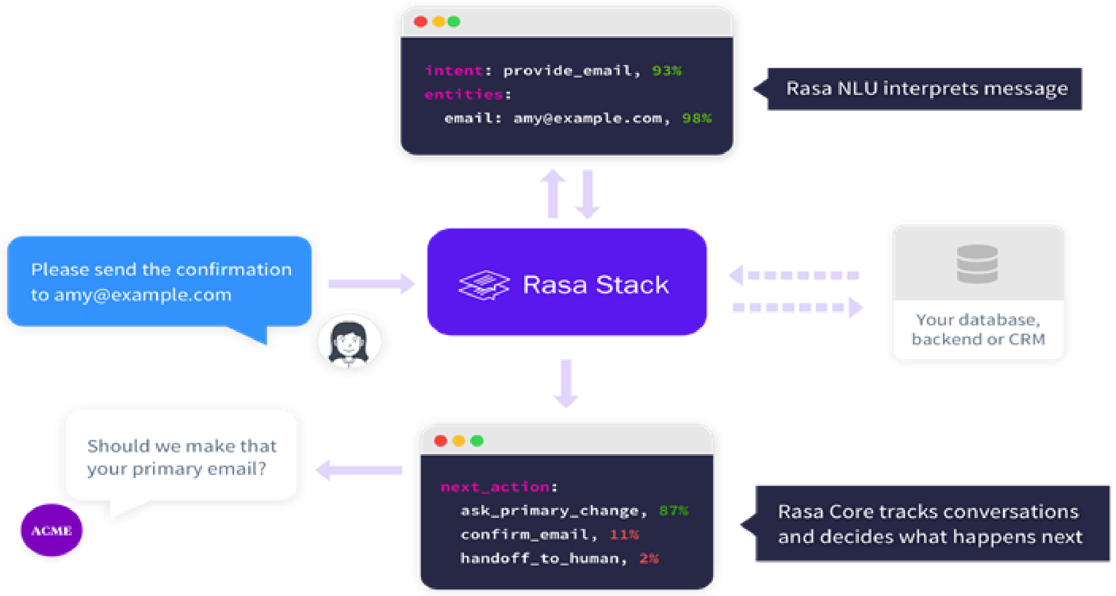
The newest revolution in AI is the ChatGPT [1]. This is one of the current state-of-the-art for chatbots in the year 2023. However, it was deemed not relevant for this internship because ChatGPT does not allow finetuning. Although GPT-models allow finetuning, they are only available in English and the chatbot should be able to understand German[1] .

At the beginning of the internship, research was done to determine whether Rasa would be the best use for the chatbot [2]. Rasa, also a state-of-the-art method, is an open-source platform. Additionally, Rasa is used inside the industry for variation of cases and companies. Rasa has the advantages of learning on its own while used, can be finetuned, and can be easily integrated, since it is open source. The basis of Rasa is nature language understanding (NLU), nature language generation (NLG) and dialogue management. With this combination, it is a good choice for the project because it can be finetuned for the specific case, and it is often used in the industry [3].

Starting the internship there was no prior knowledge of any chatbot framework. As a result, the student of the internship needed to gain knowledge about the topic. They did this by watching YouTube videos from the official Rasa channel [4]. After gaining some basic insight, a demo project was started. In this project, the student used the demo chatbot provided from Rasa. The installation of the demo chatbot was done using the command “rasa init”. Rasa provides videos on how to get started with the demo bot. While following the videos, the student changed the language of the demo chatbot from English to German.

2.2 First chatbot version

After the first few demo examples the student started to follow the demo videos but changing the training data to vocabulary that was relevant for the project. With the vocabulary the bot learns the NLU and it is trained with the NLU model. The other part of the bot is the core model. The core model selects the answers for the chatbot to the user. Figure 1 shows the architecture of Rasa.



```
intent: provide_email, 93%
entities:
  email: amy@example.com, 98%
```

Please send the confirmation to amy@example.com

```
next_action:
  ask_primary_change, 87%
  confirm_email, 11%
  handoff_to_human, 2%
```

Should we make that your primary email?

Rasa NLU interprets message
Rasa Stack
Your database, backend or CRM
ACME
Rasa Core tracks conversations and decides what happens next

Figure 1 shows the Rasa architecture and the relation between core and NLU model. Graphic is taken from [5]

2.2.1 NLU model

The NLU in Rasa works with different NLP techniques which can be configured in a YAML file. This YAML file is called pipeline. The NLU model labels utterance to an intent, for example *Hello* as greet and *I want to do an onboarding* as motivation etc. Besides the intents, it also predicts entities which are the variables for the script (hyperscaler, infra-id-, sub-id and provider) and the values of the variables as variables. The first group represents the content of the variables that are filling out the YAML file, while the second group is used to get the help for these variables.

The components of the NLU pipeline vary from traditional NLP techniques to ML. These components are split into four categories: Tokenizer, Featurizers, intent classifier and entity extractor [3]. From each category there are several types. In this report, only the ones that were used for the chatbot are explained, for more information [3].

For the Tokenizer the Whitespace Tokenizer is used. It separates text with the whitespaces into tokens. For the Featurizers several different kinds were used. The RegexFeaturizer uses regular expressions from the training data to extract words that fit into them. With the LexicalSyntacticFeaturizer lexical and syntactic features are created to support entity extraction. The CountVectorsFeaturizer creates a bag-of-word representation from the user response. This is used to help the intent and response classification. It can be used for analyzing either a word or a n-gram. In this report both variations are used, as can be seen in. The last Featurizer is the LanguageModelFeaturizer. That means pretrained language models are used to create a vector representation. In this report BERT is used [6].

Intent classification and entity extraction can done with two different components. However, in this case, a single component called Dual Intent Entity Transformer (DIET) classifier is used, which is a multi-task architecture based on a transformer [7]. The transformer is used for both tasks. Furthermore, for entity recognition a Conditional Random Field is used for the prediction after the transformer. The intent labels from the transformer are represented as a single semantic vector space. Then, a dot product is applied to the semantic vector space to maximize the similarity with the target label [3].

2.2.2 Core model

The other part of the chatbot is the core model, which consists of the policies that determine how to respond to a message from a user. These polices are trained on stories, which are created manually in the beginning and later real-life stories are used. Each policy uses the data differently. With the rule policy, rules are applied. By this means the rules are to enforce always the same answer to the same message, which ensures stability. The memorization policy compares the current stories with old ones, and the predicted answer is the answer from a previous story only if there is a 100 percent match. Finally, the TED policy, which is a transformer policy, uses the stories to learn what to answer. The TED

Policy has a similar architecture to the DIET classifier, which makes the chatbot more flexible. To determine which policy is used, priority ranking is applied. First, the rule policy is checked, if there is no answer then the memorization policy is used. Lastly the TED policy is applied. Figure 2 shows the relationship of the different components to each other.

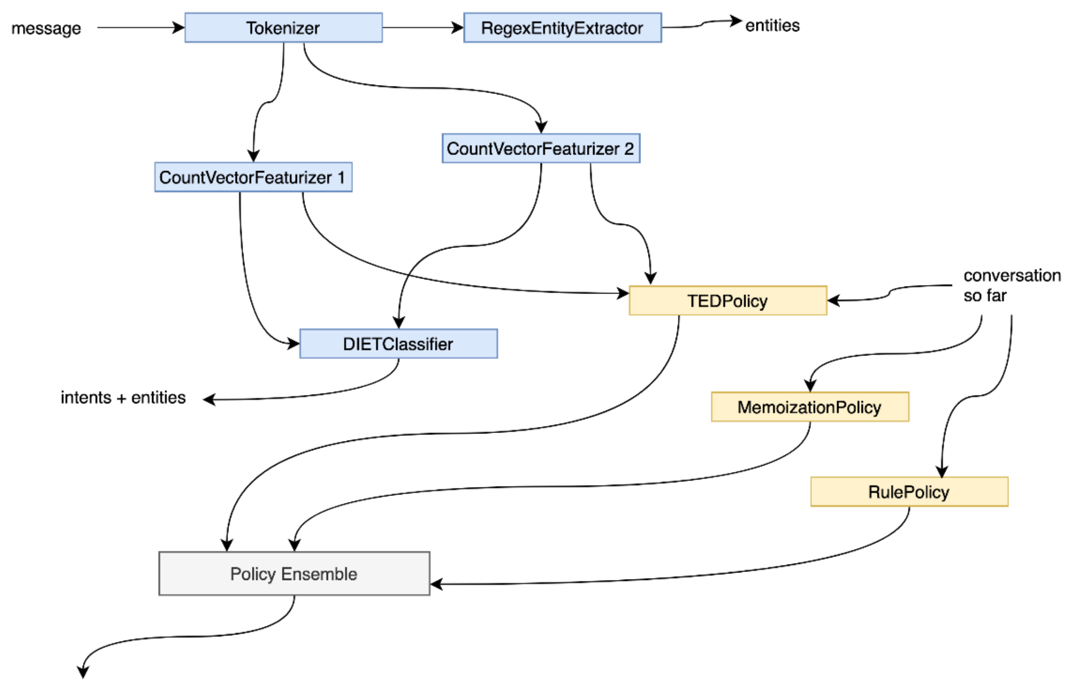


Figure 2 shows how the different components can be connected in a Rasa chatbot. Graphic is taken from [8]

2.2.3 Action server

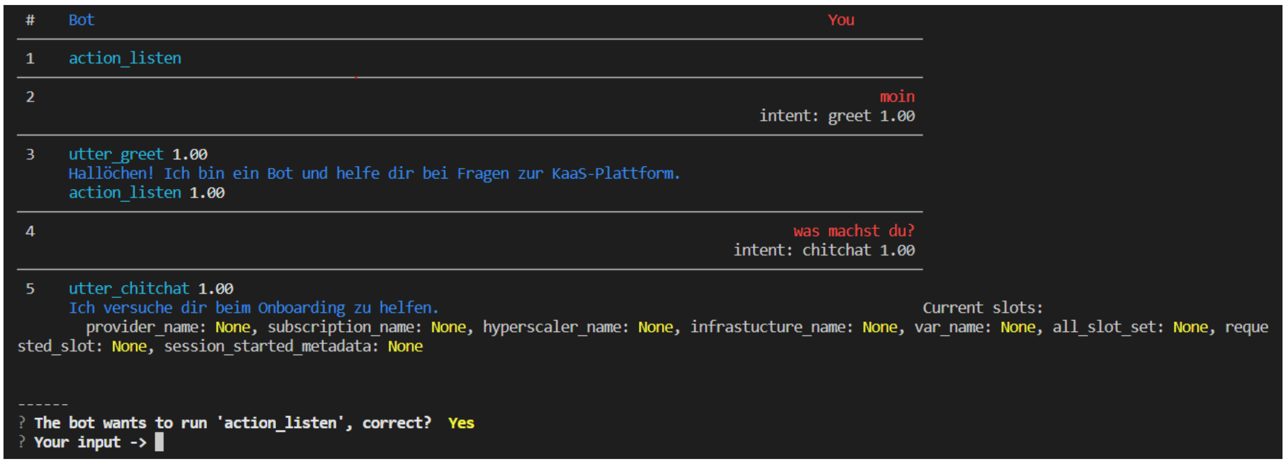
The action server allows custom actions for the bot. That can be a custom response but also any other actions that can be scripted. The first action that was done, was the validation of the variables. Another action is to check, if the subscription id for example has to a certain length, character and hyphen in certain places. Furthermore, the validation of the provider's name and hyperscaler is to check if they have a certain length. With that the validation for the infrastructure-id is to see if the fit in the range for the responding hyperscaler. Also, the validation checks if the id is already used or not.

Next to the validation there is an action that checks if all variables have a value before the summary of the variables is presented to the user. If not, the missing variable is asked again. If so, a YAML file with the variables is made. For that a template YAML file with the need structure is used and filled in with the new values for the variables. That is done also with a custom action. Another custom action is to reset all values of the variables after the YAML file is made, for a fresh start. The last custom action is to answer questions asked for the variables. The answer to that is a link to the intern documentation.

2.2.4 Training

The first training of the chatbot was done with all the described components but without the pretrained language models. Pretrained models slow down the chatbot and they are more complicated to integrate. In the beginning, the focus was to get a first easy version to see how it behaves with real-life data.

The training was done in this state with a command called "Rasa interactive". That command allows you to work with the trained chatbot and create new training data. An example of it is shown in Figure 3. That is done by playing a user and being able to correct the prediction of the chatbot, which allows it to use it as debugging when the chatbot reacts wrong to user input. There it shows if either the intent of the message is wrongly predicted or if the answer is wrongly selected from the policy. Such, allows accurate adjustment in the training data. With this method the chatbot was trained until no scenario could been found where it failed. With that the testing with real-life data started.



```
#    Bot                                                                                You
-------------------------------------------------------------------------------------------------
1    action_listen
-------------------------------------------------------------------------------------------------
2                                                                                          moin
                                                                              intent: greet 1.00
-------------------------------------------------------------------------------------------------
3    utter_greet 1.00
     Hallöchen! Ich bin ein Bot und helfe dir bei Fragen zur KaaS-Plattform.
     action_listen 1.00
-------------------------------------------------------------------------------------------------
4                                                                                 was machst du?
                                                                           intent: chitchat 1.00
-------------------------------------------------------------------------------------------------
5    utter_chitchat 1.00
     Ich versuche dir beim Onboarding zu helfen.                                                   Current slots:
       provider_name: None, subscription_name: None, hyperscaler_name: None, infrastucture_name: None, var_name: None, all_slot_set: None, reque
sted_slot: None, session_started_metadata: None


------
? The bot wants to run 'action_listen', correct?  Yes
? Your input ->
```

Figure 3: Rasa interactive with the following chitchat dialogue: U= User, C= Chatbot: U: "Hello" C: "Hello. I am a bot, and I am helping you with question about KaaS." U:" What are you doing?" C:"I am trying to help you with the Onboarding."

2.3 Interface of the chatbot

For the testing an interface is needed. The first way to do the interface was to use Rasalit [9], which is a specific version of Streamlit [10]. Streamlit is a visual web tool [10]. However, there were difficulties installing it. During the installation, Streamlit automatically downloads the newest version from Rasa. But, the dependencies from Rasa are incompatible with the dependencies with Rasalit. This made it not possible to use. Hence, different options were looked up on GitHub. One option found is an interface for chatbots called Chatroom [11]. Here the index.html was adjusted to German and the welcome message was changed fitting to the bot. Other modifications were the colors and the name of the chatbot. The chatbot is called Erna. Another challenge was to make the file accessible as a download, which was solved by adding a download directory and adjusted paths.

2.4 Deployment

After the interface was done, the chatbot had to be deployed. The first deployment was done with Docker [12] and *Docker Compose up* to get an overview of how the different applications have to be connected. For example, the YAML file is created by an action server and has to be shared with the web server for a download.

After successfully deploying in Docker, the Kubernetes deployment was started. Since the student had only small previous knowledge of Kubernetes was, coworkers from the team helped with the deployment. The first step was to explain the dependencies between the different applications. The different applications are a database, the Rasa model, the action server and the webserver. The workflow is as follows: The webserver gets the message which is put into the Rasa model. Later, the Rasa model gives all the information to the database for saving. Next, the Rasa model processes the input from the webserver and either sends back a message to the web server or activates an action from the action server. With that the action from the action server is then send back to the Rasa model which sends it back to the web server. For more complexity, the action server has to send the made YAML file to the webserver, so the file can be downloaded.

After the explanation of the dependencies, the deployments were created for the different application, along with the necessary Secrets and Ingress. This allows the different applications to communicate with each other, and the web server can be accessed via a link. With the chatbot now publicly available, user data was downloaded and evaluated daily. If new information was extracted from the data base, the information was used to update the chatbot. That provided the student with additional learning opportunities for Kubernetes and database management.

2.5 First evaluation and updating

After the deployment, the student asked their colleagues to test the chatbot to get some feedback. The feedback included the suggestion that the chatbot should be able to engage in chit chat. Additionally, the feedback gave insight of mislabeled intents and responses and suggested additional features to the action server. One of the additional features for the action servers was to see if the infrastructure id from the user had been already used. To achieve this, access to a website where the ids are documented was required. The ids from the website are compared to the input id from the user. With that it ensures that there is no id used double.

The chit chat was added with a response selector, which works like the DIET classifier. The response selector handles everything that is classified as chit chat and distinguishes between different chit chat intents. A chitchat is for example to ask what's up, if it is a chatbot or what languages it speaks.

In the first version of the chatbot, the user was asked to give feedback after a file was created. The feedback was about the satisfaction of using the chatbot. The form of the feedback was asked in the form of stars. Where one star was the lowest ranking and five stars was the highest possible ranking. After talking to Cristian, the form to give feedback changed. The new feedback was a google form in which 10 questions are asked from a System Usability Scale (SUS) template [13]. From the template the language changed from English to German and changed the word “system” to the word “bot”. That gave a better methodical view on how satisfied the users are with the chatbot. These updates were integrated in the chatbot first locally and then updated on the Kubernetes cluster.

2.6 Finetuning

With the insight of the real-life data, the finetuning of the NLU and core model were started. The testing was done by comparing different pipelines where one parameter was changed, and all components started on the same random seed. The training was done with different amounts of training data to see the generalization of the data. The testing after training was done on hand-made testing stories. In the end the results were saved in a F1-score graph.

For the NLU model the investigation into pretrained models started. One that was giving good results was the BERT model from Rasa called LaBSE [14]. LaBSE has the advantage of being trained on 12 different languages, including English and German. In an IT environment, English words are often used for technical terms. Hence, it is good that LaBSE can also classify the mix between the two languages. The theoretical benefit from LaBSE was also seen in the testing results when compared against a BERT model trained solely on German and the non-use of BERT at all. However, when updating the model to Kubernetes, the model crashed when BERT was involved. After some research and trying out different ways, no solution could be found. So, no pretrained language model could be used.

Additionally, different variations of the DIET classifier and response selector were tested. With finetuning, the DIET classifier made entity recognition without any mistakes on the test data, as shown in graphic 6.1.1 DIET classification, which is a confusion matrix shows. The graphic shows that all entity were correct recognized, since only numbers are shown in the diagonal. Based on this result, it seems that the DIET classifier does not need improvement for the entity recognition.

For the intent and response classification the finetuning resulted in a confusion matrix with a few single misclassifications, as shown graphics 6.1.2 Intent classification and 6.1.3 Response classification. In these confusion matrices, the perfect diagonal was not achieved, as some ones are outside of the diagonal, which correspond to misclassification. These results show that the classification was successful most of the time, but there is still room for improvement. Overall, the testing for the NLU model was a success because the entity recognition was done without mistakes, and intent and response classification had only a few minor mistakes.

For the core model, different pipelines for the Ted policies were compared, as it is the only one with machine learning included. The first step was to look with Rasa interactive how the model reacts to stories where it failed. This enabled to add more stories with the wanted response from what the Ted policy could learn. However, the Ted policy still did not learn. Therefore, the batch size was decreased to 16 while enhancing the training epochs to 350. This resulted in learning for the Ted policy, and some finetuning with learning rate and transformer size were done. In the end the finetuning improved the core model, but there is still room for improvement. Five responses out of twenty were not correct classified at all, as be seen in 6.1.4 Stories classification. That may result through the misbalanced classes. To address this, the classes should be tried to be more balanced in the future, and adapt the hyperparameter more to the imbalanced class problem.

After all this finetuning the colleagues tested the updated chatbot and fill out the SUS form. Ten persons filled out the form. The result of the SUS was an average score of 74. According to Sauro [13], that score has higher perceived usability than 70% of other products tested. It can also be translated to a B-. Hence, that is a satisfying result but it leaves room for improvement. Improvements can be made by detecting a wider range of detecting provider's name and offering a different format for answering questions.

4 Evaluation

The first learning goal of applying AI with real-life data was successful. After the first testing of the chatbot the errors identified. With that new knowledge from the real-life data the chatbot could been improved. Also, theoretic knowledge from the Master's program was applied on the real-life case. The knowledge helped to solve the problem with the Ted policy. The chatbot was developed in Rasa where the student had no prior experience with Rasa. With that the second learning goal of using state-of-art was accomplished. Now the student knows most of Rasa functions. Also, when the student needed help for very specific Rasa questions, the student started to ask for help in the community of Rasa. Previously the student did not ask for help in online communities. During the learning process the student learned more about the function of transformers. Additionally, during the research the student gained knowledge about the theoretic background of ChatGPT.

The deployment of the chatbot the third learning goal, to integrate a state-of-the-art method into a functional workflow, was achieved. This allowed the student to learn how products are deployed in the industry. During the deployment process, the student learned a lot, about security, folder structure and to use general paths. In more detail, the student learned about how to use Docker Compose and translates its structure to Kubernetes. During the deployment process, the student had to explain their structure to colleagues and realized that they tend to explain too complicatedly. They plan to work to improve this skill in the future. To enable the deployment, the student needed to create a web interface, which was an opportunity to deepen their knowledge. Also, they had to use a SQL database, which they only had a little bit of a theoretic background. But they were then able to apply that knowledge and deepen the knowledge.

Overall, the student learned how to communicate in a company. With that the student's skill when to ask for help improved. Also, during the internship search skill did improve, as the student needed to find answers to specific questions. Furthermore, the student got some insight into how different developers work together on a big project. With all these new impressions they know now that they want to work in developing after the student's master. With that the last learning goal, get to know the everyday work life inside the industry was reached.

During the first two weeks of the internship, the student did literature research about which platform to use. After they decided to use Rasa, the student started with the chatbot. During this time, the student was done with the first version of the chatbot. With that they were two weeks ahead of the schedule. After these two weeks, the web interface was done for a week. Here the student scheduled less time for it. In the fourth and the beginning of the fifth week the student did the deployment. Here there was a 3-day schedule. So that process took longer than expected. With the deployment done the first round of testing was done. With the feedback from the testing the structure of the chatbot was changed and the finetuning of the component started. That was done until the last week of the internship started. Last week the chatbot was tested again and the documentation started. The chatbot was developed successfully.

5 Conclusion

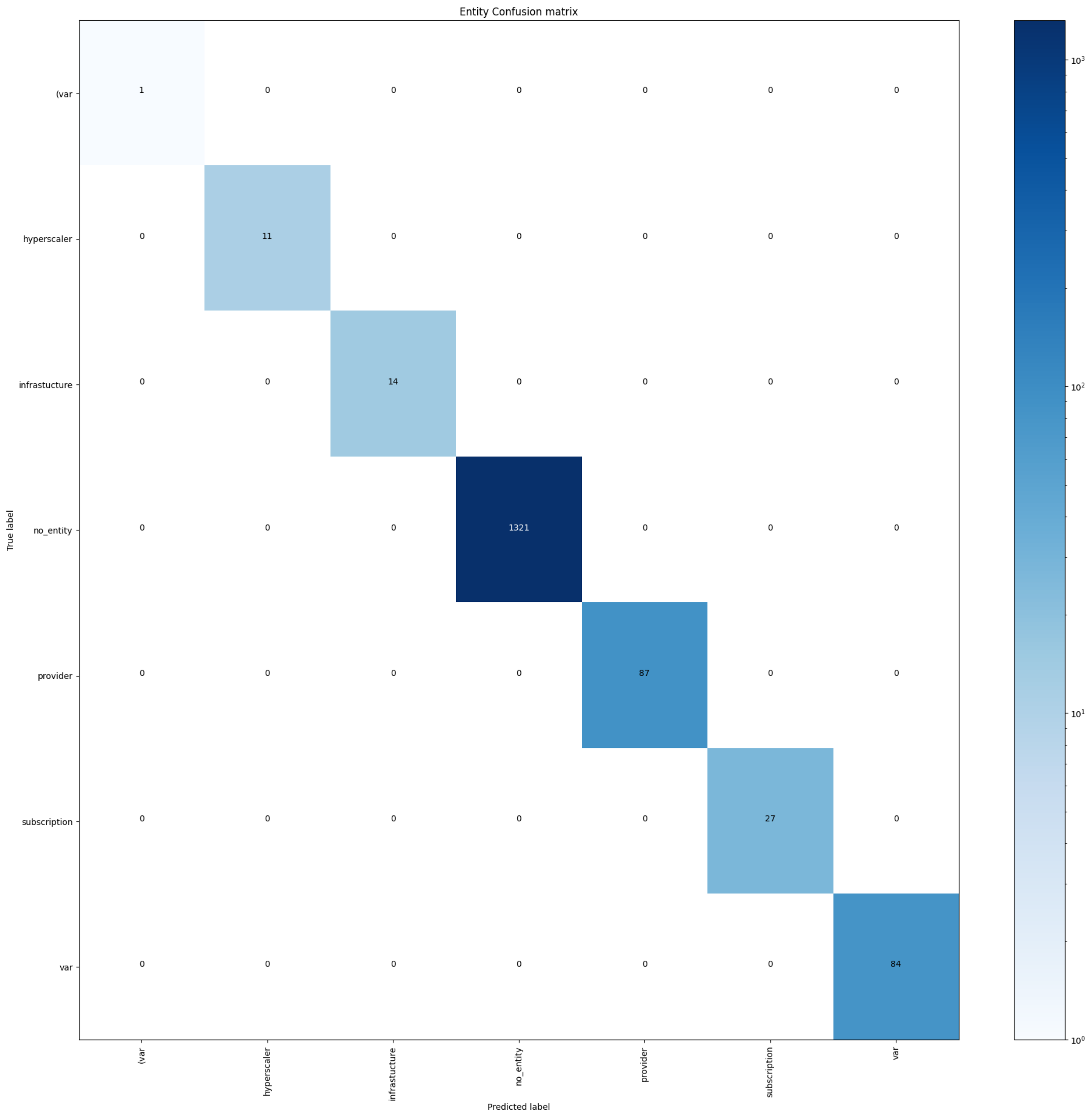
In the internship, a chatbot to help to get variables for a Kubernetes script was successful developed and deployed. The evaluation of the chatbot showed a good NLU model and core model with room for improvement. The user feedback was satisfying but with room for improvement, as it translates to the grade B-. To improve the chatbot a wide range of provider names should be classified and a different format for answering question may be used. Also, for the core model the imbalanced class problem has to be addressed.

The learning goals were achieved and new skills were learned by the student. The new skills include improved research skills, communication in a company and how and when to ask for help. Also, the student enjoyed the internship. With the schedule time table, the student was mostly successful, although sometimes the student was ahead of the schedule and sometimes behind. During the internship, the student learned that the deployment process takes a lot of time and would adjust their approach in the future. Specifically, the student would start the testing the chatbot with real users earlier than how it was done in the internship.

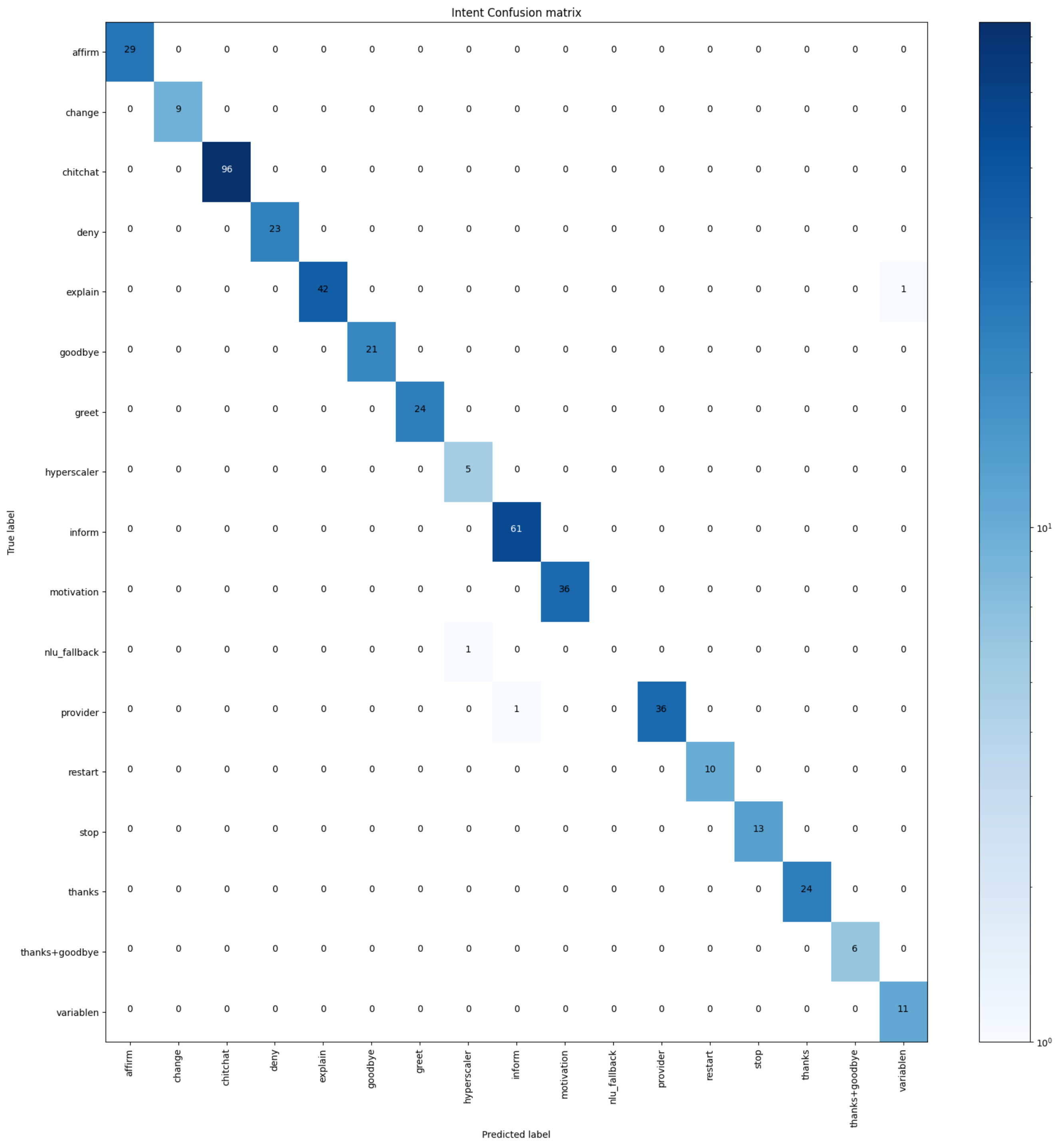
6 Appendix

6.1 Confusion matrix

6.1.1 DIET classification

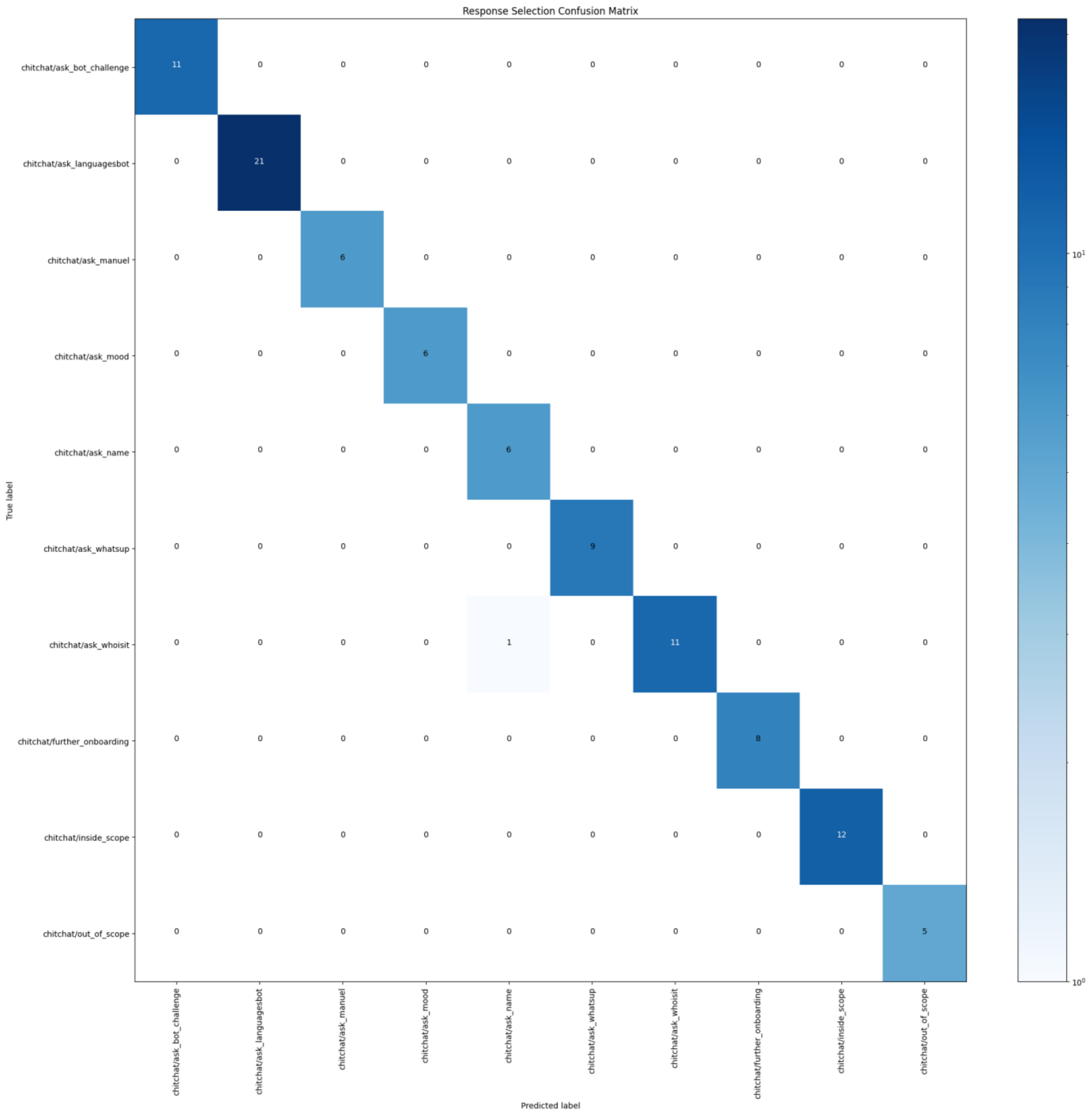


6.1.2 Intent classification

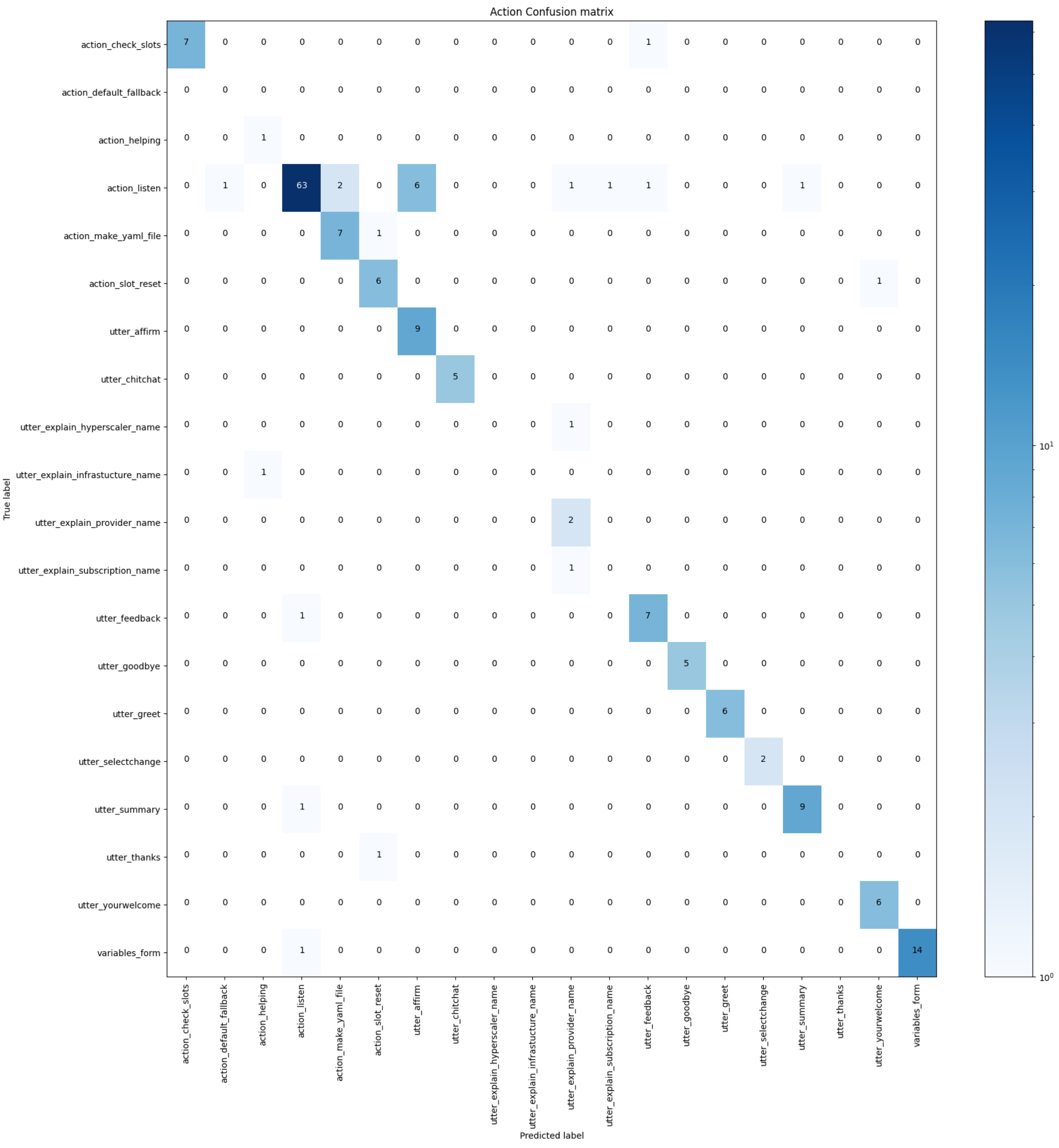


True label \ Predicted label	affirm	change	chitchat	deny	explain	goodbye	greet	hyperscaler	inform	motivation	nlu_fallback	provider	restart	stop	thanks	thanks+goodbye	variablen
affirm	29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
change	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
chitchat	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
deny	0	0	0	23	0	0	0	0	0	0	0	0	0	0	0	0	0
explain	0	0	0	0	42	0	0	0	0	0	0	0	0	0	0	0	1
goodbye	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0
greet	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0
hyperscaler	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
inform	0	0	0	0	0	0	0	0	61	0	0	0	0	0	0	0	0
motivation	0	0	0	0	0	0	0	0	0	36	0	0	0	0	0	0	0
nlu_fallback	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
provider	0	0	0	0	0	0	0	0	1	0	0	36	0	0	0	0	0
restart	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0
stop	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0
thanks	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0	0
thanks+goodbye	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0
variablen	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11

6.1.3 Response classification



6.1.4 Stories classification



	action_check_slots	action_default_fallback	action_helping	action_listen	action_make_yaml_file	action_slot_reset	utter_affirm	utter_chitchat	utter_explain_hyperscaler_name	utter_explain_infrastucture_name	utter_explain_provider_name	utter_explain_subscription_name	utter_feedback	utter_goodbye	utter_greet	utter_selectchange	utter_summary	utter_thanks	utter_yourwelcome	variables_form
action_check_slots	7	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
action_default_fallback	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
action_helping	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
action_listen	0	1	0	63	2	0	6	0	0	0	1	1	1	0	0	0	1	0	0	0
action_make_yaml_file	0	0	0	0	7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
action_slot_reset	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	1	0
utter_affirm	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0
utter_chitchat	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0
utter_explain_hyperscaler_name	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
utter_explain_infrastucture_name	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
utter_explain_provider_name	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
utter_explain_subscription_name	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
utter_feedback	0	0	0	1	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0
utter_goodbye	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
utter_greet	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0
utter_selectchange	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
utter_summary	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0
utter_thanks	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
utter_yourwelcome	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0
variables_form	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14

7 References:

[1] OpenAI, “ChatGPT.” https://openai.com/blog/chatgpt/ (accessed Jan. 25, 2023).

[2] Rasa Technologies GmbH, “Rasa.” https://rasa.com/ (accessed Jan. 25, 2023).

[3] Rasa Technologies GmbH, “Rasa componets.” https://rasa.com/docs/rasa/components/ (accessed Apr. 25, 2023).

[4] Rasa Technologies GmbH, “Rasa Youtube channel .” https://www.youtube.com/@RasaHQ (accessed May 09, 2023).

[5] M. Pethani, “Rasa architecture,” Nov. 07, 2019. https://chatbotslife.com/making-of-chatbot-using-rasa-nlu-rasa-core-part-1-7138c438581f (accessed May 09, 2023).

[6] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 2019.

[7] T. Bunk, D. Varshneya, V. Vlasov, and A. Nichol, “DIET: Lightweight Language Understanding for Dialogue Systems,” *CoRR*, vol. abs/2004.09936, 2020.

[8] Rasa Technologies GmbH, “Rasa custom graph,” May 06, 2023. https://rasa.com/docs/rasa/custom-graph-components/ (accessed May 10, 2023).

[9] Rasa Technologies GmbH, “Rasalit.” https://github.com/RasaHQ/rasalit (accessed May 10, 2023).

[10] Snowflake Inc, “Streamlit.” https://streamlit.io/ (accessed May 10, 2023).

[11] hotzenklotz., “Chatroom.” https://github.com/scalableminds/chatroom (accessed Apr. 25, 2023).

[12] Docker Inc., “Docker.” https://www.docker.com/ (accessed May 12, 2023).

[13] Jeff Sauro, “Measuring Usability with the System Usability Scale (SUS),” 2011. https://measuringu.com/sus/ (accessed Apr. 25, 2023).

[14] F. Feng, Y. Yang, D. Cer, N. Arivazhagan, and W. Wang, “Language-agnostic BERT Sentence Embedding,” in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2022. doi: 10.18653/v1/2022.acl-long.62.